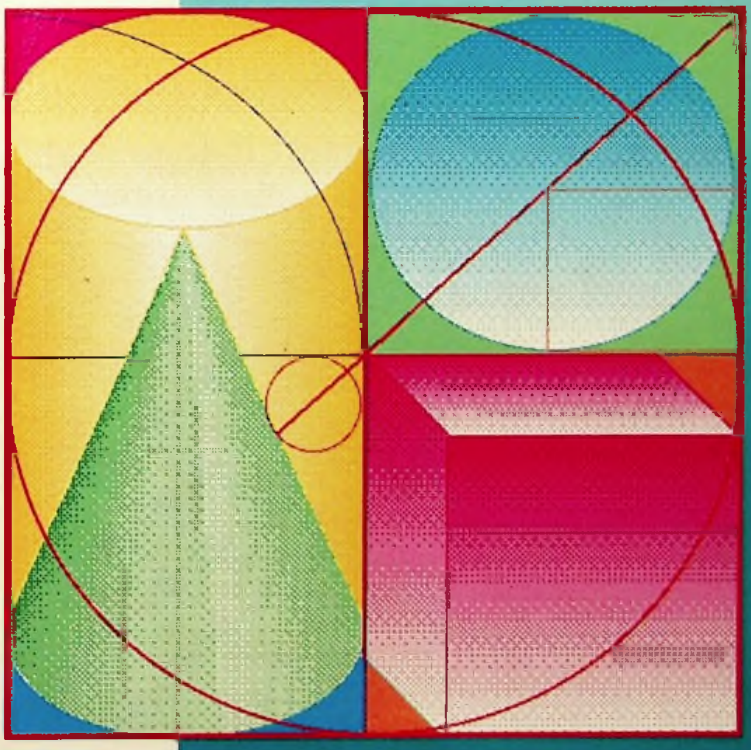
Apple® IIc Memory Expansion Card Reference Manual

APDA #A2G0047



Apple IIc Memory Expansion Card Reference Manual

This package contains the following items:

1	Manual:	*Apple IIc Memory Expansion Card Reference Manual*	A2G0047

If you have any questions, please call APDA at 1-800-282-2732.

Apple® II

Apple® IIc Memory Expansion Card Technical Reference

APPLE COMPUTER, INC.

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This excepuon does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

© Apple Computer, Inc., 1987
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple, the Apple logo, and ProDOS are registered trademarks of Apple Computer, Inc.

ProFile is a trademark of Apple Computer, Inc.

Simultaneously published in the United States and Canada.

Contents

Figures and Tables iv

Preface vii

About the Apple IIc Memory Expansion Card vii
About this manual viii

Chapter 1 **Introduction to the Apple IIc Memory Expansion Card 1**

How the memory expansion card works 2
Using the memory expansion card as a storage device 2
- ProDOS 3
- Pascal (version 3.1 and later) 3
- DOS 3.3 4

Using the memory expansion card as a startup device 4

Chapter 2 **Hardware 5**

The RAM 6
The gate array 7
- RAM refresh 7
- RAM addressing 7

The PAL 7
The bus driver 8
The connector 8

Chapter 3 **Overview of the Firmware Interface 9**

How the interface works 10
Making a ProDOS/Pascal call 10
- Command parameters zero-page write 11
- ProDOS/Pascal call 11

Making a Smartport call 12
- Smartport location 12

Smartport call 13
An overview of calls 16
STATUS ($00) 17
READ BLOCK ($01) 17
WRITE BLOCK ($02) 18
FORMAT ($03) 18
CONTROL ($04) 19
READ ($08) 19
WRITE ($09) 20
Summary of error codes 20

Appendix A **Firmware Map 23**

Glossary 25

Figures and tables

Figure 2-1: Memory expansion card component layout 6
Figure 2-2: Memory expansion card connector 8
Figure 3-1: Smartport call formats 14-15

Table 3-1: Microprocessor register states 12
Table A-1: Main side ROM map 23
Table A-2: Auxiliary side ROM map 24

Preface

About the Apple IIc Memory Expansion Card

The Apple® IIc Memory Expansion Card is an accessory card that provides your Apple IIc with up to one megabyte (1Mb) of additional program and data storage. The basic memory expansion card provides 256K of RAM and can be expanded in 256K steps to 512K, 768K, or 1024K by your authorized Apple dealer, even after initial installation.

The memory expansion card is supported by a firmware interface that permits card access to be executed through either the Smartport or the operating system entry points. A brief discussion of the commands used by this firmware interface has been included in this manual. For more information on the Smartport and operating system interfaces, see the *Apple IIc Technical Reference*.

Important Apple strongly recommends that the memory expansion card be treated as a block storage device, like a RAM disk, rather than as direct-access RAM. The Smartport routines written into the new Apple IIc ROM make using the memory expansion card as a block-type device straightforward and simple.

About this manual

This reference manual is intended as a desktop resource for programmers, developers, and enthusiasts using the many features available to them with the addition of the memory expansion card. If you are unfamiliar with the basic principles of memory and memory-mapped I/O in the Apple IIc, refer to the sections on memory operations in the *Apple IIc Technical Reference*

Chapter 1

Introduction to the Apple IIc Memory Expansion Card

The Apple IIc Memory Expansion Card is a high-volume (up to 1Mb) RAM device for program and data storage. Although it is a RAM device, it acts like a **block device** rather than like direct-access memory. This means that programs can be loaded into the card's RAM, but they must be moved into the Apple IIc's main memory in order to be executed.

A **block device** executes I/O operations by grouping data into bundles, called **blocks**. A block may be made up of virtually any number of bytes, but in the Apple II family a standard block is 512 bytes.

How the memory expansion card works

The Apple IIc Memory Expansion Card works like a very fast block-type storage device. Programs and data are loaded into the card's RAM by calling assembly-language routines in either the operating system or the **Smartport.** These routines provide the means to read, write, and manipulate the memory expansion card as a block device.

A **call** is a set of software instructions that pass all of the data necessary for an assembly-language routine to be executed.

To use these routines from a program, you must issue a **call.** Calls are briefly described in Chapter 3.

The memory expansion card supports the ProDOS®, Pascal (version 1.3 and later), and DOS 3.3 operating systems.

Using the memory expansion card as a storage device

The memory expansion card provides RAM storage in block-type device format. You can get the most out of the memory expansion card by using it as a **RAM disk.** Because it is a RAM device, it operates much more quickly than an actual disk drive.

A **RAM disk** is a range of RAM that the CPU treats like a disk drive. Because the RAM doesn't need mechanical read/write heads and drive motors, it is much faster than an actual disk drive.

Important Remember that the memory expansion card is a RAM device; a loss of power will cause the contents of the card to be erased.

Before you can use the memory expansion card, it must be formatted by the memory expansion firmware. The firmware first checks address $BF00 to identify the operating system running on the Apple IIc; the memory expansion card is then formatted for that operating system.

Different operating systems cannot share the memory expansion card. To use the card with an operating system other than the one that runs the first access of the card, you must reformat the card. The following paragraphs describe how ProDOS®, Pascal, and DOS 3.3 set up the memory expansion card and how to reformat the card (if necessary) for use with each system.

ProDOS

ProDOS loads $BF00 with $4C and formats the memory expansion card as a volume named /RAM4. To reformat the memory expansion card for use with ProDOS, use the ProDOS Filer and format the card as a volume named /RAM4.

Pascal (version 1.3 and later)

Pascal loads $BF00 with $00 and formats the memory expansion card as a volume named RAM4. To reformat the memory expansion card for use with Pascal, use the Pascal Filer and format the card as a volume named RAM4. Earlier versions of Pascal (prior to 1.3) require a special driver, similar to the ProFile™ driver.

DOS 3.3

DOS 3.3 does not support the ProDOS block device protocol. To set up the memory expansion card for use by DOS 3.3, type `IN#4`. This will format the card as DISK VOLUME 254 , with a maximum capacity of 400K.

Because DOS is altered to allow the memory expansion card to intercept read/write to track sector (RWTS) calls, the INIT command is disabled. With the INIT command disabled, disks cannot be initialized by DOS.

Using the memory expansion card as a startup device

To use the memory expansion card as a startup device, you must first format it for use with an operating system using the system utilities. Once the card is properly formatted, you may load it with boot code. You cannot use the memory expansion card as a startup device from a cold start. The memory expansion card is the first slot checked in the Apple IIc's startup procedure.

Once you have loaded the memory expansion card with boot code, you can execute a warm reset of the Apple IIc by pressing Control-Open Apple-Reset, or by issuing a reset command from the operating system. Startup of the Apple IIc then proceeds normally.

Chapter 2

Hardware

The Apple IIc Memory Expansion Card is expandable from its basic size of 256K in 256K steps to 512k, 768K, or 1024K. Each step consists of eight 256Kx1 RAM integrated circuits (ICs) installed in sockets across the face of the card. In its maximum configuration (1Mb), the card contains 32 RAM ICs. The card also contains a gate array, a PAL, and a bus driver.

The RAM

The memory expansion card can contain a maximum of 32 RAM ICs in 4 groups of 8 ICs each. The groups are arranged in rows across the face of the card, as shown in Figure 2-1.

The memory expansion card has a 20-bit address register and a 1-byte data register. The data register is used to read or write data at the location pointed to by the address register. Because the address register automatically increments after each access of the data register, do not use index operations that cause false reads when you are accessing the data register. The memory expansion card's registers are mapped into main memory at the following locations:

- $C0C0 low byte of the address register
- $C0C1 middle byte of the address register
- $C0C2 high nibble (D3 to D0) of the address register
- $C0C3 data register

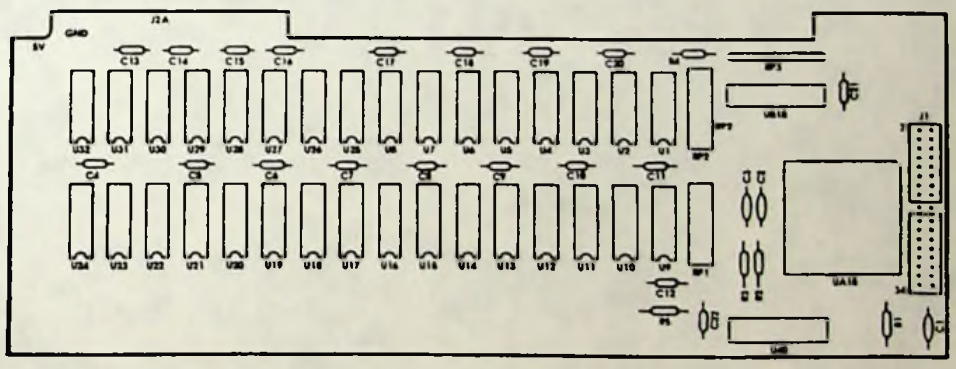


Figure 2-1
Memory expansion card component layout

The gate array

CMOS stands for Conductive Metal-Oxide Semiconductor.

The gate array is a custom **CMOS** IC that provides the standard Apple IIc I/O bus electrical interface. It provides four basic functions for the memory expansion card:

- RAM refresh
- RAM addressing
- timing
- address decoding

RAM refresh

Each 256K of dynamic RAM is refreshed every 4 milliseconds. A refresh cycle is performed during alternating phase 1 clock cycles. During a refresh cycle, one 256K set of RAM is refreshed; the remaining sets, if present, are refreshed during subsequent refresh cycles. This technique helps to minimize power consumption and does not conflict with CPU access time. A 256K RAM refresh is completed in about 2 milliseconds.

RAM addressing

The RAM ICs are addressed through an auto increment 20-bit address pointer. Before accessing the RAM, the address pointer must be set to the first location to be read or written. After each RAM access the address pointer is automatically incremented to the next location. The address pointer must be loaded in the following order: low byte, middle byte, high byte.

The PAL

The programmable array logic (PAL) is a standard 16L8A-2CN type PAL IC. It provides on-board control logic for the RAM array.

The bus driver

The bus driver is a 74LHCT245 TTL IC.

The connector

The memory expansion card is connected to the Apple IIc's main logic board by a 34-pin card-edge connector. Figure 2-3 is a pin-out diagram of the memory expansion card hard connector.

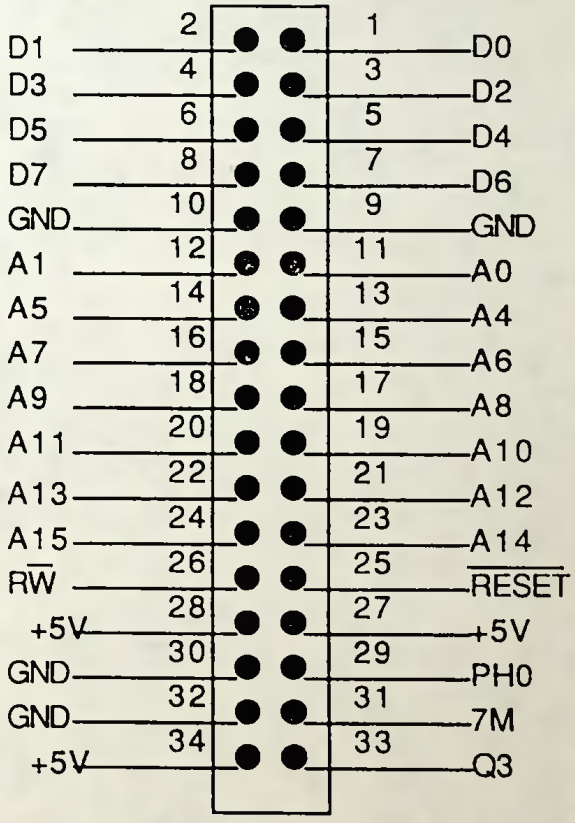


Figure 2-2
Memory expansion card connector

Chapter 3

Overview of the Firmware Interface

Programs access and control the Apple IIc Memory Expansion Card through the firmware interface. This chapter provides you with a brief look at that interface, including instructions on issuing calls and a summary of the command set.

How the interface works

When a program running on the Apple IIc needs to access the memory expansion card, it must issue a call to the firmware interface. A call is made up of a **command number**, a **pointer**, and a **parameter list.** Calls to the firmware interface are issued to either the Smartport or the operating system running on the Apple IIc.

Calls to the ProDOS or Pascal operating systems are processed according to the ProDOS block device protocol.

Once a call is issued to the firmware interface, the Apple IIc turns control over to the entry point until the command has executed. When it has completed its execution, the firmware interface returns control to the Apple IIc and program execution resumes at the statement immediately following the call.

Making a ProDOS/Pascal call

Calls to the ProDOS or Pascal operating systems are executed according to the ProDOS block device protocol. The ProDOS block device protocol uses two basic stages:

- Command parameters zero-page write
- ProDOS/Pascal call

Command parameters zero-page write

Prior to issuing a firmware call, the operating system passes a set of command parameters to the firmware interface. These parameters are passed in zero-page locations $42–$47, as follows:

- $42 command number
- $43 slot number (slot 4 for the memory expansion card)
- $44–$45 buffer pointer
- $46–$47 block number

The command number is the firmware command number. The slot number is set in bits 4 and 5. The buffer pointer indicates the start of a 512-byte data buffer. The block number is the address of the target block on the memory expansion card. If the command is not a read/write operation, no block number is required.

ProDOS/Pascal call

The second stage is the actual call. A ProDOS/Pascal call is coded in assembly language as a JSR (jump to subroutine) to the entry point.

You can calculate the entry point by reading $C4FF. $C4FF contains the low byte of the firmware interface entry address.

A flag is a variable whose value (1 or 0) indicates the state of a specific parameter.

When the firmware has completed the operation requested by the call, it sets certain **flags** in the microprocessor's Status register and accumulator (A register). The state of these flags depends on whether the ProDOS/Pascal call was successfully completed. Table 3-1 defines the state of the microprocessor flags for both successful and unsuccessful calls.

Table 3-1
Microprocessor register states

Bit	Successful call	Unsuccessful call
N	x	x
Z	x	x
C	0	1
D	0	0
V	x	x
I	unchanged	unchanged
B	x	x
Xreg	x	x
Yreg	x	x
A	0	error code
PC	JSR+3	JSR+3
SP	unchanged	unchanged

❖ *Note:* x is undefined unless index information is returned in X and Y.

Making a Smartport call

A Smartport call is performed in two basic stages:

- Determine the location of the entry point for Smartport
- Issue a Smartport call

Smartport location

Before you issue a call to the Smartport, it's a good idea to make sure that the Smartport is present on the Apple IIc. This step helps ensure compatibility between software and future hardware developments.

To locate the Smartport, search for the following bytes:

- $C401 = $20
- $C403 = $00
- $C405 = $03
- $C407 = $00

Smartport call

A Smartport call is coded in assembly language as a JSR to the Smartport entry point (DISPATCH), followed by the 1-byte command number, followed by the 2-byte command parameter list pointer.

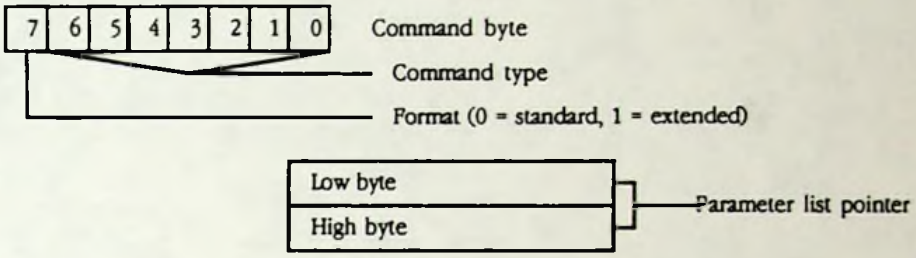
You can calculate the DISPATCH address as follows:

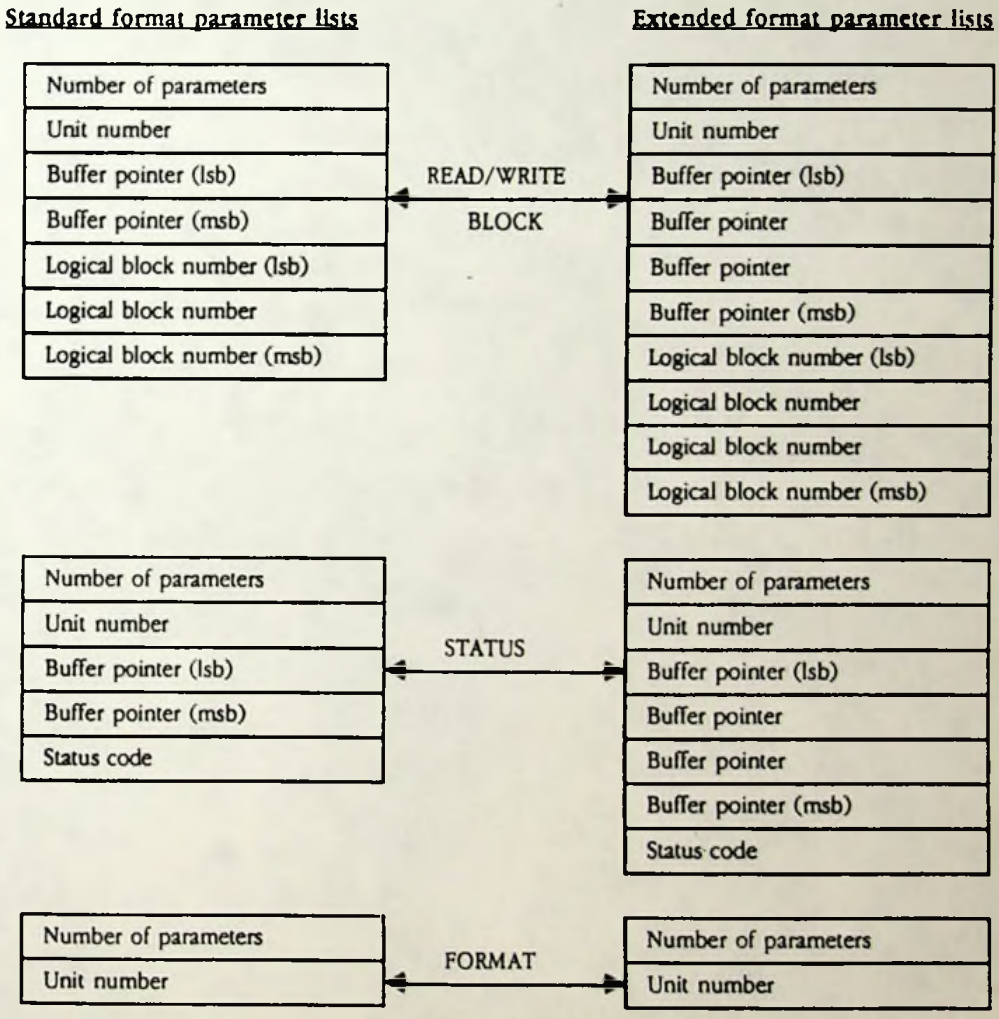
$C400 + (*C4FF*) + 3 where *C4FF* is the value of the byte located at address C4FF.

The following is an example of a Smartport call:

```
PCCALL    JSR DISPATCH    ;Call SMP entry point and
                          ;dispatcher

          DFB CMDNUM      ;SMP command number

          DW  CMDLIST     ;Command parameter list
                          ;pointer

          BCS ERROR       ;Carry is set on an error
```

Extended-format calls include a 24-bit (or greater) address field in the parameter list. The larger address field is used for data buffer and read/write addresses. Figure 3-1 diagrams the Smartport call formats.


7 6 5 4 3 2 1 0
Command byte
Command type
Format (0 = standard, 1 = extended)
Low byte
High byte
Parameter list pointer


Standard format parameter lists
Extended format parameter lists
Number of parameters
Unit number
Buffer pointer (lsb)
Buffer pointer (msb)
Logical block number (lsb)
Logical block number
Logical block number (msb)
READ/WRITE
BLOCK
Number of parameters
Unit number
Buffer pointer (lsb)
Buffer pointer
Buffer pointer
Buffer pointer (msb)
Logical block number (lsb)
Logical block number
Logical block number
Logical block number (msb)
Number of parameters
Unit number
Buffer pointer (lsb)
Buffer pointer (msb)
Status code
STATUS
Number of parameters
Unit number
Buffer pointer (lsb)
Buffer pointer
Buffer pointer
Buffer pointer (msb)
Status code
Number of parameters
Unit number
FORMAT
Number of parameters
Unit number

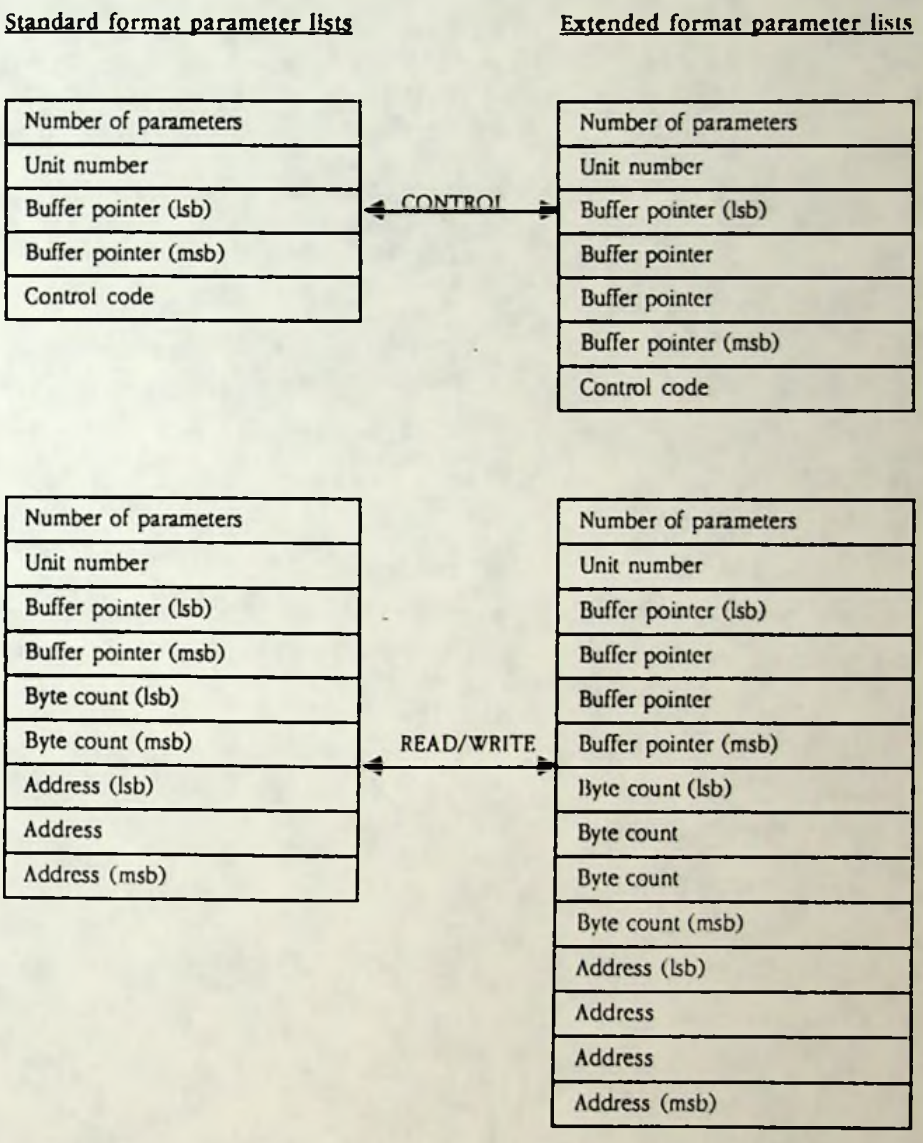


Figure 3-1
Smartport call formats

When the Smartport has completed the operation requested by the call, it sets certain flags in the microprocessor's Status register and accumulator (A register). There are two differences between the states of the microprocessor register flags after a Smartport call and those given for ProDOS/Pascal calls (Table 3-1):

- The X and Y registers (Xreg and Yreg) are set to the number of bytes of data transferred in successful card-to-CPU transfers; unsuccessful transfers remain undefined.
- The PC register is set to JSR+5 for extended-format Smartport calls.

An overview of calls

The following sections briefly describe each interface command. Each definition contains the following information:

- command name: the name of the firmware command
- command number: the hexadecimal number of the firmware command
- description: the purpose of the command
- parameter list: the command parameter list required by or for the command, by byte number (given in parentheses)

STATUS ($00)

Description

The STATUS command returns information about the memory expansion card . The information returned by this command is determined by the status code parameter.

On return from a STATUS call, the microprocessor X and Y registers are set to indicate the number of bytes transferred to the Apple IIc by the command. The X register is set to the low byte of the count, and the Y register is set to the high byte.

Parameter list

- *parameter count (0):* 1 byte, set to 3 for this command
- *unit number (1):* 1 byte
- *status list pointer (2,3)*:* 1 word, data buffer start address
- *status code (4):* 1 byte

*These bytes are processed as a single parameter.

READ BLOCK ($01)

Description

The READ BLOCK command reads one 512-byte block from the target device specified in the unit number parameter. The block read by this command is written into bank $00 RAM at the address specified in the data buffer pointer.

Parameter list

- *parameter count (0):* 1 byte, set to 3 for this command
- *unit number (1):* 1 byte
- *data buffer pointer (2,3)*:* 1 word, data buffer address
- *block number (4,5,6)*:* 3 bytes, logical address of target block

*These bytes are processed as a single parameter.

WRITE BLOCK ($02)

Description

The WRITE BLOCK command writes one 512-byte block to the target device specified in the unit number parameter. The block written by this command is read from bank $00 RAM at the address specified in the data buffer pointer.

Parameter list

- *parameter count (0):* 1 byte, set to 3 for this command
- *unit number (1):* 1 byte
- *data buffer pointer (2,3)*:* 1 word, data buffer address
- *block number (4,5,6)*:* 3 bytes, logical address of target block

*These bytes are processed as a single parameter.

FORMAT ($03)

Description

The FORMAT command prepares all the blocks on the device specified in the unit number parameter for read/write use. This command is for use on block-type devices only.

Important Operating system-specific information, such as bit maps and catalogs, is *not* prepared by the FORMAT command.

Parameter list

- *parameter count (0):* 1 byte, set to 1 for this command
- *unit number (1):* 1 byte

CONTROL ($04)

Description

The CONTROL command writes a 512-byte control block to the device specified in the unit number parameter. The control block written to the target device contains a control command and a list of data required by the target device to execute that command.

Parameter list

- *parameter count (0):* 1 byte, set to 3 for this command
- *unit number (1):* 1 byte
- *control list pointer (2,3)*:* 1 word, data buffer low and high address
- *control code (4):* 1 byte

*These bytes are processed as a single parameter.

READ ($08)

Description

The READ command reads a specified number of bytes from the target device specified in the unit number parameter. The bytes read by this command are written into bank $00 RAM, beginning at the address specified in the data buffer pointer. The number of bytes to be read is specified in the byte count parameter.

Parameter list

- *parameter count (0):* 1 byte, set to 4 for this command
- *unit number (1):* 1 byte
- *data buffer pointer (2,3)*:* 1 word, data buffer address
- *byte count (4,5)*:* 2 bytes, number of bytes to read
- *address pointer (6,7,8)*:* 3 bytes, device-specific parameter

*These bytes are processed as a single parameter.

WRITE ($09)

Description

The WRITE command writes a specified number of bytes to the target device specified in the unit number parameter. The bytes written by this command are read from bank $00 RAM, beginning at the address specified in the data buffer pointer. The number of bytes to be written is specified in the byte count parameter.

Parameter list

- *parameter count (0):* 1 byte, set to 4 for this command
- *unit number (1):* 1 byte
- *data buffer pointer (2,3)*:* 1 word, data buffer address
- *byte count (4,5)*:* 2 bytes, number of bytes to read
- *address pointer (6,7,8)*:* 3 bytes, device-specific parameter

*These bytes are processed as a single parameter.

Summary of error codes

Following is a summary of error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the Status register of the microprocessor) is cleared (0), and the accumulator (the A register) contains 0's. If the call was unsuccessful, the C flag is set (1), and the A register contains the error code.

$00		No error occurred.
$01	BadCmd	A nonexistent command was issued. Check the command number.
$04	BadPCnt	This identifies a bad parameter count. The parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.

$06	BusErr	A communications error exists between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources; make sure the cable is properly shielded.
$11	BadUnit	Unit number $00 was used in a call other than STATUS, CONTROL, or INIT.
$21	BadCtl	The CONTROL or STATUS code is not supported by the device.
$22	BadCtlParm	The CONTROL parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.
$27	IOError	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective. Make sure the device is operating correctly.
$28	NoDrive	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.
$2B	NoWrite	The medium in the device is write-protected.
$2D	BadBlock	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided vs. double-sided disk, for example).
$2F	OffLine	The device is off-line or there is no disk in the drive. Check the cables and connections; make sure the medium is present in the drive, and that the drive is functioning correctly.

$30–$3F	DevSpec	This identifies errors that differ from device to device. See the technical manual for the device in question for details.
$40–$4F	Reserved	Reserved for future expansion.
$50–$7F	NonFatal	A device-specific soft error occurred. The operation completed successfully, but some exception condition was detected. See the technical manual for the device in question for details.

Appendix A

Firmware Map

Appendix A is a map of the Apple IIc's main ROM. For a listing of the IIc's main ROM, see Appendix I in the *Apple IIc Technical Reference*.

Table A-1
Main side ROM map

C100	Serial port
C200	Communications port
C300	80-column routines
C400	Memory expansion card: boot code/entry point for ProDOS/entry point for DOS
C500–C58D	UniDisk 3.5 routines
C58E	Miscellaneous
C600	Disk II routines
C700–C77F	Mouse entry points
C780	ROM switch routines
C7FC	Last screen hole
C800	Interrupt routines
C900	Paddle patch; mini-assembler
CA00	Mini-Assembler; step and trace
CB00	Scroll routines
CC00	pasinvert; picky; showcur; update; escape
CD00	Video routines
CE00	Video routines
CF00	Video routines; miscellaneous
D000–F7FF	Applesoft BASIC
F800–FFFF	RESET vectors

Table A-2
Auxiliary side ROM map

C100	Mouse interrupt handler; ACIA interrupt handler
C200	continue; ACIA routines
C300	ACIA routines; moveaux; xfer; memory test
C400	continue; switch test
C500–C57F	diagnostics
C780	ROM switch routines
C800–C87F	Miscellaneous
C880–CFFF	UniDisk 3.5 routines
D000	Command processor for serial and communications ports
D100	continue
D200–D249	Mouse BASIC routines
D24A–D3FF	Empty
D400	basilin; hex-to-dec
D4A9–D52B	Zero page test
D500–D52B	Miscellaneous
D52C–D5FF	Empty
D600–D700	Mouse routines
D800	execute; xdiag; xstatus; pstat0; pcn41; pstatus; xread; xwrite; prdblk; pwrblk; pread
D900	pread continue; pwrite
DA00	dospatch; dosconv; stattbl; parmtbl; undtbl; testsize; format
DB00	fmpas; fmdos; makecat; cattbl; procut
DC00	doscut; pascut
DCF0–DCFF	diagnostics (orged at $DD00)
DD00	stattest; addresstest; rollovertest; addbustest; cleartest; fulltest
DE00	continue; computed; pass; fail; miscellaneous
DF00	Print; messages
E000–FFF9	Empty

Glossary

accumulator: The register in the microprocessor where most computations are performed.

address: (1) A number that specifies the location of a single **byte** of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from $0000 to $FFFF (in hexadecimal). (2) In data transmission, a code for a specific terminal. Multiple terminals on one communication line, for example, must have unique addresses.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form that's easier to understand than machine language itself. Each assembly language instruction produces one machine-language instruction.

bit: A contraction of *binary digit.* The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

block: A unit of data. A standard block is 512 bytes in size. Non-standard blocks must be specially defined.

block device: A block device executes I/O operations by grouping data into bundles, called **blocks.** A block may be made up of virtually any number of bytes, but in the Apple II family a standard block is 512 bytes in size.

buffer: A "holding area" of the computer's RAM where information can be stored on a temporary basis (buffered). Buffering is often done when data is being transferred between two devices operating at different communication rates.

bus: An electrical or electronic connection between devices. The devices connected by the bus are said to be resident on the bus, and may be as small as ICs or as large as mainframe computers. A bus provides a means to send the same data, signals, or voltages (for power supply buses) to more than one device across a single carrier (wire, fiber-optic cable, and so on).

byte: A unit of data consisting of eight contiguous bits numbered from 0 to 7. Bit 0 of a byte is called the **most significant bit (MSB),** while bit 7 is called the **least significant bit (LSB).**

call: A set of software instructions that pass all of the data required by the firmware interface to execute a **command.** A call contains a **command number,** a **pointer,** and a **parameter list.**

central processing unit (CPU): The Apple II computer; specifically, the microprocessor and its supporting hardware exclusive of any peripheral cards or devices, RAM, and ROM.

code: (1) A number or symbol that corresponds to a set of parameters or instructions. (2) The statements that make up a program.

command: An instruction that causes the target device to perform a specific operation. A command consists of a **command number**, a **pointer**, and a **parameter list.**

command number: A hexadecimal number that corresponds to a specific firmware command. Each firmware command has a unique command number.

command register: A location in a device controller that stores control information. Compare **data register.**

control code: A hexadecimal number that corressponds to a device-specfic control function.

CPU: See **central processing unit.**

data register: A location in a peripheral device controller that stores data. Compare **command register.**

device: A hardware unit, such as a computer, a disk drive, or a peripheral card.

error code: A number or other symbol representing a type of error.

field: A specific set of data that is related. A field is always defined by its size, given in bits or bytes. A field usually has a name.

firmware: Programs stored permanently in read-only memory (ROM). Such programs (for example, the Smartport) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory.

flag: A variable whose value (1 or 0) indicates the state of a specific parameter.

hexadecimal: The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. For example, the decimal numbers 0, 1, 2, 3, 4, ... 8, 9, 10, 11, ... 15, 16, 17 would be shown in hexadecimal notation as 00, 01, 02, 03, 04, ... 08, 09, 0A, 0B, ... 0F, 10, 11. Hexadecimal numbers are easier for people to read and understand than binary numbers, and they can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a sequence of four bits. Hexadecimal numbers are usually preceded by a dollar sign ($).

initialize: To set to a predetermined starting state or value.

input/output (I/O): The process by which information is transferred to and from the computer and peripheral devices.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

interface: The rules of interaction between two devices or programs; the electrical, procedural, and functional conventions that govern the exchange of data and control signals between devices.

interrupt: An electronic "attention getter"; a signal sent to the microprocessor that is intended to force the microprocessor to stop its current activity and accept input from the device that sent the interrupt.

least significant bit (LSB): The rightmost bit of a binary number; bit 0. The least significant bit contributes the smallest quantity to the value of the number. Compare **most significant bit (MSB).**

main memory: The part of a computer's volatile (non-permanent) memory that is directly accessible to the microprocessor. Auxiliary memory, such as that contained on peripheral cards, is mapped into main memory to allow the microprocessor to access it.

most significant bit (MSB): The leftmost bit of a **byte;** bit 7. The most significant bit contributes the largest quantity to the value of the number. Compare **least significant bit (LSB).**

null: Any character or character code that has no meaning to the operating system or program interpreting it.

parameter: Any of a set of characteristics whose value or condition determines the operation of a program or device.

parameter list: The list of characteristics whose value or condition determines the precise execution of a call.

pointer: A data item that provides the memory address of some other data item; a pointer contains the address of some other data.

protocol: A formal set of rules for sending and receiving data on a communication line.

RAM disk: A range of RAM that the CPU treats as a disk drive. Because the RAM doesn't need mechanical read/write heads and drive motors, it is much faster than an actual disk drive.

register: A memory location of a specific size in which each bit (or byte, depending on the size and design of the register) has some meaning to a microprocessor, program, or "smart" IC. IC registers are sometimes internal to the IC.

Smartport: A set of assembly language routines used in the Apple II family for performing block device I/O.

stack: A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

unit number: The number assigned to a device based on the position of a hardware strap, jumper or switch on the device itself.

X register: One of the two index registers in the 6502 microprocessor.

Y register: One of the two index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple II family of computers, also called page zero. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter™ Plus. POSTSCRIPT™, the LaserWriter's page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.